CHAPTER 1. SETTING UP THE APACHE HTTP WEB SERVER

1.1. INTRODUCTION TO THE APACHE HTTP WEB SERVER

A *web server* is a network service that serves content to a client over the web. This typically means web pages, but any other documents can be served as well. Web servers are also known as HTTP servers, as they use the *hypertext transport protocol* (**HTTP**).

The **Apache HTTP Server**, **httpd**, is an open source web server developed by the Apache Software Foundation.

If you are upgrading from a previous release of Red Hat Enterprise Linux, you will need to update the **httpd** service configuration accordingly. This section reviews some of the newly added features, and guides you through the update of prior configuration files.

1.1.1. Notable changes in the Apache HTTP Server

The **Apache HTTP Server**, has been updated from version 2.4.6 to version 2.4.37 between RHEL 7 and RHEL 8. This updated version includes several new features, but maintains backwards compatibility with the RHEL 7 version at the level of configuration and Application Binary Interface (ABI) of external modules.

New features include:

- **HTTP/2** support is now provided by the **mod_http2** package, which is a part of the **httpd** module.
- systemd socket activation is supported. See **httpd.socket(8)** man page for more details.
- Multiple new modules have been added:
  - **mod_proxy_hcheck** - a proxy health-check module
  - **mod_proxy_uwsgi** - a Web Server Gateway Interface (WSGI) proxy
  - **mod_proxy_fdpass** - provides support for the passing the socket of the client to another process
  - **mod_cache_socache** - an HTTP cache using, for example, memcache backend
  - **mod_md** - an ACME protocol SSL/TLS certificate service
- The following modules now load by default:
  - **mod_request**
  - **mod_macro**
  - **mod_watchdog**
- A new subpackage, **httpd-filesystem**, has been added, which contains the basic directory layout for the **Apache HTTP Server** including the correct permissions for the directories.
- Instantiated service support, **httpd@.service** has been introduced. See the **httpd.service** man page for more information.

- A new **httpd-init.service** replaces the **%post script** to create a self-signed **mod_ssl** key pair.
- Automated TLS certificate provisioning and renewal using the Automatic Certificate Management Environment (ACME) protocol is now supported with the **mod_md** package (for use with certificate providers such as **Let's Encrypt**).
- The **Apache HTTP Server** now supports loading TLS certificates and private keys from hardware security tokens directly from **PKCS#11** modules. As a result, a **mod_ssl** configuration can now use **PKCS#11** URLs to identify the TLS private key, and, optionally, the TLS certificate in the **SSLCertificateKeyFile** and **SSLCertificateFile** directives.
- A new **ListenFree** directive in the **/etc/httpd/conf/httpd.conf** file is now supported. Similarly to the **Listen** directive, **ListenFree** provides information about IP addresses, ports, or IP address-and-port combinations that the server listens to. However, with **ListenFree**, the **IP_FREEBIND** socket option is enabled by default. Hence, **httpd** is allowed to bind to a nonlocal IP address or to an IP address that does not exist yet. This allows **httpd** to listen on a socket without requiring the underlying network interface or the specified dynamic IP address to be up at the time when **httpd** is trying to bind to it.

  Note that the **ListenFree** directive is currently available only in RHEL 8.

  For more details on **ListenFree**, see the following table:

  Table 1.1. ListenFree directive's syntax, status, and modules

Syntax	Status	Modules
ListenFree [IP-address:]portnumber [protocol]	MPM	event, worker, prefork, mpm_winnt, mpm_netware, mpmt_os2

Other notable changes include:

- The following modules have been removed:
  - **mod_file_cache**
  - **mod_nss**
    Use **mod_ssl** as a replacement. For details about migrating from **mod_nss**, see Section 1.10, "Exporting a private key and certificates from an NSS database to use them in an Apache web server configuration".
  - **mod_perl**
- The default type of the DBM authentication database used by the **Apache HTTP Server** in RHEL 8 has been changed from **SDBM** to **db5**.
- The **mod_wsgi** module for the **Apache HTTP Server** has been updated to Python 3. WSGI applications are now supported only with Python 3, and must be migrated from Python 2.
- The multi-processing module (MPM) configured by default with the **Apache HTTP Server** has changed from a multi-process, forked model (known as **prefork**) to a high-performance multi-threaded model, **event**.

Any third-party modules that are not thread-safe need to be replaced or removed. To change the configured MPM, edit the **/etc/httpd/conf.modules.d/00-mpm.conf** file. See the **httpd.service(8)** man page for more information.

- The minimum UID and GID allowed for users by suEXEC are now 1000 and 500, respectively (previously 100 and 100).
- The **/etc/sysconfig/httpd** file is no longer a supported interface for setting environment variables for the **httpd** service. The **httpd.service(8)** man page has been added for the systemd service.
- Stopping the **httpd** service now uses a "graceful stop" by default.
- The **mod_auth_kerb** module has been replaced by the **mod_auth_gssapi** module.

1.1.2. Updating the configuration

To update the configuration files from the **Apache HTTP Server** version used in Red Hat Enterprise Linux 7, choose one of the following options:

- If **/etc/sysconfig/httpd** is used to set environment variables, create a systemd drop-in file instead.
- If any third-party modules are used, ensure they are compatible with a threaded MPM.
- If suexec is used, ensure user and group IDs meet the new minimums.

You can check the configuration for possible errors by using the following command:

```
# apachectl configtest
Syntax OK
```

1.2. THE APACHE CONFIGURATION FILES

When the **httpd** service is started, by default, it reads the configuration from locations that are listed in Table 1.2, "The httpd service configuration files" .

Table 1.2. The httpd service configuration files

Path	Description
/etc/httpd/conf/httpd.conf	The main configuration file.
/etc/httpd/conf.d/	An auxiliary directory for configuration files that are included in the main configuration file.
/etc/httpd/conf.modules.d/	An auxiliary directory for configuration files which load installed dynamic modules packaged in Red Hat Enterprise Linux. In the default configuration, these configuration files are processed first.

Although, the default configuration is suitable for most situations, you can use also other configuration options. For any changes to take effect, restart the web server first. See Section 1.3, "Managing the httpd service" for more information on how to restart the **httpd** service.

To check the configuration for possible errors, type the following at a shell prompt:

```
# apachectl configtest
Syntax OK
```

To make the recovery from mistakes easier, make a copy of the original file before editing it.

1.3. MANAGING THE HTTPD SERVICE

This section describes hot to start, stop, and restart the **httpd** service.

Prerequisites

- The Apache HTTP Server is installed.

Procedure

- To start the **httpd** service, enter:

```
# systemctl start httpd
```

- To stop the **httpd** service, enter:

```
# systemctl stop httpd
```

- To restart the **httpd** service, enter:

```
# systemctl restart httpd
```

1.4. SETTING UP A SINGLE-INSTANCE APACHE HTTP SERVER

This section describes how to set up a single-instance Apache HTTP Server to serve static HTML content.

Follow the procedure in this section if the web server should provide the same content for all domains associated with the server. If you want to provide different content for different domains, set up name-based virtual hosts. For details, see Section 1.5, "Configuring Apache name-based virtual hosts" .

Procedure

1. Install the **httpd** package:

```
# yum install httpd
```

2. Open the TCP port **80** in the local firewall:

```
# firewall-cmd --permanent --add-port=80/tcp
# firewall-cmd --reload
```

3. Enable and start the **httpd** service:

```
# systemctl enable --now httpd
```

4. Optional: Add HTML files to the /**var**/**www**/**html**/ directory.

NOTE

When adding content to /**var**/**www**/**html**/, files and directories must be readable by the user under which **httpd** runs by default. The content owner can be the either the **root** user and **root** user group, or another user or group of the administrator's choice. If the content owner is the **root** user and **root** user group, the files must be readable by other users. The SELinux context for all the files and directories must be **httpd_sys_content_t**, which is applied by default to all content within the /**var**/**www** directory.

Verification steps

- Connect with a web browser to **http://*server_IP_or_host_name*/**.
  If the /**var**/**www**/**html**/ directory is empty or does not contain an **index.html** or **index.htm** file, Apache displays the **Red Hat Enterprise Linux Test Page**. If /**var**/**www**/**html**/ contains HTML files with a different name, you can load them by entering the URL to that file, such as **http://*server_IP_or_host_name*/*example.html***.

Additional resources

- For further details about configuring Apache and adapting the service to your environment, refer to the Apache manual. For details about installing the manual, see Section 1.8, "Installing the Apache HTTP Server manual".
- For details about using or adjusting the **httpd systemd** service, see the **httpd.service(8)** man page.

1.5. CONFIGURING APACHE NAME-BASED VIRTUAL HOSTS

Name-based virtual hosts enable Apache to serve different content for different domains that resolve to the IP address of the server.

The procedure in this section describes setting up a virtual host for both the **example.com** and **example.net** domain with separate document root directories. Both virtual hosts serve static HTML content.

Prerequisites

- Clients and the web server resolve the **example.com** and **example.net** domain to the IP address of the web server.
  Note that you must manually add these entries to your DNS server.

Procedure

1. Install the **httpd** package:

```
# yum install httpd
```

2. Edit the **/etc/httpd/conf/httpd.conf** file:

   a. Append the following virtual host configuration for the **example.com** domain:

```
<VirtualHost *:80>
    DocumentRoot "/var/www/example.com/"
    ServerName example.com
    CustomLog /var/log/httpd/example.com_access.log combined
    ErrorLog /var/log/httpd/example.com_error.log
</VirtualHost>
```

   These settings configure the following:

   - All settings in the **<VirtualHost *:80>** directive are specific for this virtual host.
   - **DocumentRoot** sets the path to the web content of the virtual host.
   - **ServerName** sets the domains for which this virtual host serves content.
     To set multiple domains, add the **ServerAlias** parameter to the configuration and specify the additional domains separated with a space in this parameter.
   - **CustomLog** sets the path to the access log of the virtual host.
   - **ErrorLog** sets the path to the error log of the virtual host.

NOTE

Apache uses the first virtual host found in the configuration also for requests that do not match any domain set in the **ServerName** and **ServerAlias** parameters. This also includes requests sent to the IP address of the server.

3. Append a similar virtual host configuration for the **example.net** domain:

```
<VirtualHost *:80>
    DocumentRoot "/var/www/example.net/"
    ServerName example.net
    CustomLog /var/log/httpd/example.net_access.log combined
    ErrorLog /var/log/httpd/example.net_error.log
</VirtualHost>
```

4. Create the document roots for both virtual hosts:

```
# mkdir /var/www/example.com/
# mkdir /var/www/example.net/
```

5. If you set paths in the **DocumentRoot** parameters that are not within **/var/www/**, set the **httpd_sys_content_t** context on both document roots:

```
# semanage fcontext -a -t httpd_sys_content_t "/srv/example.com(/.*)?"
# restorecon -Rv /srv/example.com/
# semanage fcontext -a -t httpd_sys_content_t "/srv/example.net(/.\*)?"
# restorecon -Rv /srv/example.net/
```

These commands set the **httpd_sys_content_t** context on the **/srv/example.com/** and **/srv/example.net/** directory.

Note that you must install the **policycoreutils-python-utils** package to run the **restorecon** command.

6. Open port **80** in the local firewall:

```
# firewall-cmd --permanent --add-port=80/tcp
# firewall-cmd --reload
```

7. Enable and start the **httpd** service:

```
# systemctl enable --now httpd
```

Verification steps

1. Create a different example file in each virtual host's document root:

```
# echo "vHost example.com" > /var/www/example.com/index.html
# echo "vHost example.net" > /var/www/example.net/index.html
```

2. Use a browser and connect to **http://example.com**. The web server shows the example file from the **example.com** virtual host.

3. Use a browser and connect to **http://example.net**. The web server shows the example file from the **example.net** virtual host.

Additional resources

- For further details about configuring Apache virtual hosts, refer to the **Virtual Hosts** documentation in the Apache manual. For details about installing the manual, see Section 1.8, "Installing the Apache HTTP Server manual".

1.6. CONFIGURING TLS ENCRYPTION ON AN APACHE HTTP SERVER

By default, Apache provides content to clients using an unencrypted HTTP connection. This section describes how to enable TLS encryption and configure frequently used encryption-related settings on an Apache HTTP Server.

Prerequisites

- The Apache HTTP Server is installed and running.

1.6.1. Adding TLS encryption to an Apache HTTP Server

This section describes how to enable TLS encryption on an Apache HTTP Server for the **example.com** domain.

Prerequisites

- The Apache HTTP Server is installed and running.

- The private key is stored in the **/etc/pki/tls/private/example.com.key** file.
  For details about creating a private key and certificate signing request (CSR), as well as how to request a certificate from a certificate authority (CA), see your CA's documentation.
  Alternatively, if your CA supports the ACME protocol, you can use the **mod_md** module to automate retrieving and provisioning TLS certificates.
- The TLS certificate is stored in the **/etc/pki/tls/private/example.com.crt** file. If you use a different path, adapt the corresponding steps of the procedure.
- The CA certificate is stored in the **/etc/pki/tls/private/ca.crt** file. If you use a different path, adapt the corresponding steps of the procedure.
- Clients and the web server resolve the host name of the server to the IP address of the web server.

Procedure

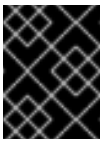
1. Install the **mod_ssl** package:

```
# dnf install mod_ssl
```

2. Edit the **/etc/httpd/conf.d/ssl.conf** file and add the following settings to the **<VirtualHost _default_:443>** directive:

   a. Set the server name:

```
ServerName example.com
```



IMPORTANT

The server name must match the entry set in the **Common Name** field of the certificate.

   b. Optional: If the certificate contains additional host names in the **Subject Alt Names** (SAN) field, you can configure **mod_ssl** to provide TLS encryption also for these host names. To configure this, add the **ServerAliases** parameter with corresponding names:

```
ServerAlias www.example.com server.example.com
```

   c. Set the paths to the private key, the server certificate, and the CA certificate:

```
SSLCertificateKeyFile "/etc/pki/tls/private/example.com.key"
SSLCertificateFile "/etc/pki/tls/certs/example.com.crt"
SSLCACertificateFile "/etc/pki/tls/certs/ca.crt"
```

3. For security reasons, configure that only the **root** user can access the private key file:

```
# chown root:root /etc/pki/tls/private/example.com.key
# chmod 600 /etc/pki/tls/private/example.com.key
```

WARNING

If the private key was accessed by unauthorized users, revoke the certificate, create a new private key, and request a new certificate. Otherwise, the TLS connection is no longer secure.

4. Open port **443** in the local firewall:

```
# firewall-cmd --permanent --add-port=443
# firewall-cmd --reload
```

5. Restart the **httpd** service:

```
# systemctl restart httpd
```

NOTE

If you protected the private key file with a password, you must enter this password each time when the **httpd** service starts.

Verification steps

- Use a browser and connect to **https://*example.com***.

Additional resources

- For further details about configuring TLS, refer to the **SSL/TLS Encryption** documentation in the Apache manual. For details about installing the manual, see Section 1.8, "Installing the Apache HTTP Server manual".

1.6.2. Setting the supported TLS protocol versions on an Apache HTTP Server

By default, the Apache HTTP Server on RHEL 8 uses the system-wide crypto policy that defines safe default values, which are also compatible with recent browsers. For example, the **DEFAULT** policy defines that only the **TLSv1.2** and **TLSv1.3** protocol versions are enabled in apache.

This section describes how to manually configure which TLS protocol versions your Apache HTTP Server supports. Follow the procedure if your environment requires to enable only specific TLS protocol versions, for example:

- If your environment requires that clients can also use the weak **TLS1** (TLSv1.0) or **TLS1.1** protocol.
- If you want to configure that Apache only supports the **TLSv1.2** or **TLSv1.3** protocol.

Prerequisites

- TLS encryption is enabled on the server as described in Section 1.6.1, "Adding TLS encryption to an Apache HTTP Server".

Procedure

1. Edit the **/etc/httpd/conf/httpd.conf** file, and add the following setting to the **<VirtualHost>** directive for which you want to set the TLS protocol version. For example, to enable only the **TLSv1.3** protocol:

```
SSLProtocol -All TLSv1.3
```

2. Restart the **httpd** service:

```
# systemctl restart httpd
```

Verification steps

1. Use the following command to verify that the server supports **TLSv1.3**:

```
# openssl s_client -connect example.com:443 -tls1_3
```

2. Use the following command to verify that the server does not support **TLSv1.2**:

```
# openssl s_client -connect example.com:443 -tls1_2
```

   If the server does not support the protocol, the command returns an error:

```
140111600609088:error:1409442E:SSL routines:ssl3_read_bytes:tlsv1 alert protocol
version:ssl/record/rec_layer_s3.c:1543:SSL alert number 70
```

3. Optional: Repeat the command for other TLS protocol versions.

Additional resources

- For further details about the system-wide crypto policy, see the **update-crypto-policies(8)** man page and Using system-wide cryptographic policies .
- For further details about the **SSLProtocol** parameter, refer to the **mod_ssl** documentation in the Apache manual. For details about installing the manual, see Section 1.8, "Installing the Apache HTTP Server manual".

1.6.3. Setting the supported ciphers on an Apache HTTP Server

By default, the Apache HTTP Server on RHEL 8 uses the system-wide crypto policy that defines safe default values, which are also compatible with recent browsers. For the list of ciphers the system-wide crypto allows, see the **/etc/crypto-policies/back-ends/openssl.config** file.

This section describes how to manually configure which ciphers your Apache HTTP Server supports. Follow the procedure if your environment requires specific ciphers.

Prerequisites

- TLS encryption is enabled on the server as described in Section 1.6.1, "Adding TLS encryption to an Apache HTTP Server".

Procedure

1. Edit the **/etc/httpd/conf/httpd.conf** file, and add the **SSLCipherSuite** parameter to the **<VirtualHost>** directive for which you want to set the TLS ciphers:

```
SSLCipherSuite
"EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH:!SHA1:!SHA256"
```

   This example enables only the **EECDH+AESGCM**, **EDH+AESGCM**, **AES256+EECDH**, and **AES256+EDH** ciphers and disables all ciphers which use the **SHA1** and **SHA256** message authentication code (MAC).

2. Restart the **httpd** service:

```
# systemctl restart httpd
```

Verification steps

1. To display the list of ciphers the Apache HTTP Server supports:

   a. Install the **nmap** package:

```
# yum install nmap
```

   b. Use the **nmap** utility to display the supported ciphers:

```
# nmap --script ssl-enum-ciphers -p 443 example.com
...
PORT    STATE SERVICE
443/tcp open  https
| ssl-enum-ciphers:
|   TLSv1.2:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (ecdh_x25519) - A
|       TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (dh 2048) - A
|       TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (ecdh_x25519) - A
...
```

Additional resources

- For further details about the system-wide crypto policy, see the **update-crypto-policies(8)** man page and Using system-wide cryptographic policies .
- For further details about the **SSLCipherSuite** parameter, refer to the **mod_ssl** documentation in the Apache manual. For details about installing the manual, see Section 1.8, “Installing the Apache HTTP Server manual”.

1.7. CONFIGURING TLS CLIENT CERTIFICATE AUTHENTICATION

Client certificate authentication enables administrators to allow only users who authenticate using a certificate to access resources on the web server. This section describes how to configure client certificate authentication for the **/var/www/html/Example/** directory.

If the Apache HTTP Server uses the TLS 1.3 protocol, certain clients require additional configuration. For example, in Firefox, set the **security.tls.enable_post_handshake_auth** parameter in the **about:config** menu to **true**. For further details, see Transport Layer Security version 1.3 in Red Hat

Enterprise Linux 8.

Prerequisites

- TLS encryption is enabled on the server as described in Section 1.6.1, "Adding TLS encryption to an Apache HTTP Server".

Procedure

1. Edit the **/etc/httpd/conf/httpd.conf** file and add the following settings to the **<VirtualHost>** directive for which you want to configure client authentication:

```
<Directory "/var/www/html/Example/">
  SSLVerifyClient require
</Directory>
```

   The **SSLVerifyClient require** setting defines that the server must successfully validate the client certificate before the client can access the content in the **/var/www/html/Example/** directory.

2. Restart the **httpd** service:

```
# systemctl restart httpd
```

Verification steps

1. Use the **curl** utility to access the **https://example.com/Example/** URL without client authentication:

```
$ curl https://example.com/Example/
curl: (56) OpenSSL SSL_read: error:1409445C:SSL routines:ssl3_read_bytes:tlsv13 alert certificate required, errno 0
```

   The error indicates that the web server requires a client certificate authentication.

2. Pass the client private key and certificate, as well as the CA certificate to **curl** to access the same URL with client authentication:

```
$ curl --cacert ca.crt --key client.key --cert client.crt https://example.com/Example/
```

   If the request succeeds, **curl** displays the **index.html** file stored in the **/var/www/html/Example/** directory.

Additional resources

- For further details about client authentication, see the **mod_ssl Configuration How-To** documentation in the Apache manual. For details about installing the manual, see Section 1.8, "Installing the Apache HTTP Server manual".

1.8. INSTALLING THE APACHE HTTP SERVER MANUAL

This section describes how to install the Apache HTTP Server manual. This manual provides a detailed documentation of, for example:

- Configuration parameters and directives
- Performance tuning
- Authentication settings
- Modules
- Content caching
- Security tips
- Configuring TLS encryption

After installing the manual, you can display it using a web browser.

Prerequisites

- The Apache HTTP Server is installed and running.

Procedure

1. Install the **httpd-manual** package:

```
# yum install httpd-manual
```

2. Optional: By default, all clients connecting to the Apache HTTP Server can display the manual. To restrict access to a specific IP range, such as the **192.0.2.0/24** subnet, edit the **/etc/httpd/conf.d/manual.conf** file and add the **Require ip 192.0.2.0/24** setting to the **<Directory "/usr/share/httpd/manual">** directive:

```
<Directory "/usr/share/httpd/manual">
...
    Require ip 192.0.2.0/24
...
</Directory>
```

3. Restart the **httpd** service:

```
# systemctl restart httpd
```

Verification steps

1. To display the Apache HTTP Server manual, connect with a web browser to **http://*host_name_or_IP_address*/manual/**

1.9. WORKING WITH MODULES

Being a modular application, the **httpd** service is distributed along with a number of *Dynamic Shared Objects* (**DSO**s), which can be dynamically loaded or unloaded at runtime as necessary. These modules are located in the **/usr/lib64/httpd/modules/** directory.

1.9.1. Loading a module

To load a particular DSO module, use the **LoadModule** directive. Note that modules provided by a separate package often have their own configuration file in the **/etc/httpd/conf.modules.d/** directory.

Example 1.1. Loading the mod_ssl DSO

```
LoadModule ssl_module modules/mod_ssl.so
```

After loading the module, restart the web server to reload the configuration. See Section 1.3, "Managing the httpd service" for more information on how to restart the **httpd** service.

1.9.2. Writing a module

To create a new DSO module, make sure you have the **httpd-devel** package installed. To do so, enter the following command as **root**:

```
# yum install httpd-devel
```

This package contains the include files, the header files, and the **APache eXtenSion** (**apxs**) utility required to compile a module.

Once written, you can build the module with the following command:

```
# apxs -i -a -c module_name.c
```

If the build was successful, you should be able to load the module the same way as any other module that is distributed with the **Apache HTTP Server**.

1.10. EXPORTING A PRIVATE KEY AND CERTIFICATES FROM AN NSS DATABASE TO USE THEM IN AN APACHE WEB SERVER CONFIGURATION

RHEL 8 no longer provides the **mod_nss** module for the Apache web server, and Red Hat recommends using the **mod_ssl** module. If you store your private key and certificates in a Network Security Services (NSS) database, for example, because you migrated the web server from RHEL 7 to RHEL 8, follow this procedure to extract the key and certificates in Privacy Enhanced Mail (PEM) format. You can then use the files in the **mod_ssl** configuration as described in Section 1.6, "Configuring TLS encryption on an Apache HTTP Server".

This procedure assumes that the NSS database is stored in **/etc/httpd/alias/** and that you store the exported private key and certificates in the **/etc/pki/tls/** directory.

Prerequisites

- The private key, the certificate, and the certificate authority (CA) certificate are stored in an NSS database.

Procedure

1. List the certificates in the NSS database:

```
# certutil -d /etc/httpd/alias/ -L
```

```
Certificate Nickname          Trust Attributes
                              SSL,S/MIME,JAR/XPI

Example CA                    C,,
Example Server Certificate    u,u,u
```

You need the nicknames of the certificates in the next steps.

2. To extract the private key, you must temporarily export the key to a PKCS #12 file:

   a. Use the nickname of the certificate associated with the private key, to export the key to a PKCS #12 file:

   ```
   # pk12util -o /etc/pki/tls/private/export.p12 -d /etc/httpd/alias/ -n "Example Server Certificate"
   Enter password for PKCS12 file: password
   Re-enter password: password
   pk12util: PKCS12 EXPORT SUCCESSFUL
   ```

   Note that you must set a password on the PKCS #12 file. You need this password in the next step.

   b. Export the private key from the PKCS #12 file:

   ```
   # openssl pkcs12 -in /etc/pki/tls/private/export.p12 -out
   /etc/pki/tls/private/server.key -nocerts -nodes
   Enter Import Password: password
   MAC verified OK
   ```

   c. Delete the temporary PKCS #12 file:

   ```
   # rm /etc/pki/tls/private/export.p12
   ```

3. Set the permissions on **/etc/pki/tls/private/server.key** to ensure that only the **root** user can access this file:

   ```
   # chown root:root /etc/pki/tls/private/server.key
   # chmod 0600 /etc/pki/tls/private/server.key
   ```

4. Use the nickname of the server certificate in the NSS database to export the CA certificate:

   ```
   # certutil -d /etc/httpd/alias/ -L -n "Example Server Certificate" -a -o
   /etc/pki/tls/certs/server.crt
   ```

5. Set the permissions on **/etc/pki/tls/certs/server.crt** to ensure that only the **root** user can access this file:

   ```
   # chown root:root /etc/pki/tls/certs/server.crt
   # chmod 0600 /etc/pki/tls/certs/server.crt
   ```

6. Use the nickname of the CA certificate in the NSS database to export the CA certificate:

   ```
   # certutil -d /etc/httpd/alias/ -L -n "Example CA" -a -o /etc/pki/tls/certs/ca.crt
   ```

7. Follow Section 1.6, "Configuring TLS encryption on an Apache HTTP Server" to configure the Apache web server, and:
    - Set the **SSLCertificateKeyFile** parameter to **/etc/pki/tls/private/server.key**.
    - Set the **SSLCertificateFile** parameter to **/etc/pki/tls/certs/server.crt**.
    - Set the **SSLCACertificateFile** parameter to **/etc/pki/tls/certs/ca.crt**.

Additional resources

- The **certutil(1)** man page
- The **pk12util(1)** man page
- The **pkcs12(1ssl)** man page

1.11. ADDITIONAL RESOURCES

- **httpd(8)** – The manual page for the **httpd** service containing the complete list of its command-line options.
- **httpd.service(8)** – The manual page for the **httpd.service** unit file, describing how to customize and enhance the service.
- **httpd.conf(5)** – The manual page for **httpd** configuration, describing the structure and location of the **httpd** configuration files.
- **apachectl(8)** – The manual page for the **Apache HTTP Server** Control Interface.
- For information on how to configure Kerberos authentication on an Apache HTTP server, see Using GSS-Proxy for Apache httpd operation. Using Kerberos is an alternative way to enforce client authorization on an Apache HTTP Server.